KONZEPTE DER SOFTWAREVISUALISIERUNG FÜR KOMPLEXE, OBJEKTORIENTIERTE SOFTWARESYSTEME

Kapitel 1

Grundlagen der Softwarevisualisierung

Johannes Bohnet und Jürgen Döllner

Kapitel 1: Grundlagen der Softwarevisualisierung

Johannes Bohnet und Jürgen Döllner

Zusammenfassung. Das Gewinnen von Verständnis über die Struktur und das Verhalten von Softwaresystemen stellt aufgrund der meist immensen Systemgröße und der Evolution des Systems eine anspruchsvolle, zeit- und kostenintensive Aufgabe in der Softwareentwicklung und Softwarepflege dar. Softwarevisualisierung kann den Prozess der Verständnisgewinnung unterstützen, indem sie Zusammenhänge, Abhängigkeiten und Einflüsse zwischen Artefakten des Softwaresystems offen legt. Softwarevisualisierung auf der Grundlage des implementierten Systems findet Einsatz während des gesamten Entwicklungsprozesses und ist unabhängig von der Anwendungsdomäne einsetzbar. Die zu visualisierenden implementierungsbasierten Informationen können dabei aus der Quellcodebeschreibung des Softwaresystems extrahiert, während des Programmablaufs erzeugt oder über die Betrachtung der zeitlichen Entwicklung des Systems generiert werden. Optional können zusätzlich Informationen über weitere an der Softwareentwicklung beteiligten Aspekte in die Visualisierung einfließen.

1.1 Motivation

Heutige Softwaresysteme gehören mit zu den komplexesten, von Menschen erstellten Systemen. Das Verständnis von Strukturen, Abhängigkeiten, Einflüsse und Verhalten der Systeme und ihrer Komponenten steht im Mittelpunkt von Verfahren der Softwarevisualisierung. Sie verfolgen darüber hinaus das Ziel, Einsicht in die mit ihnen verbundenen Entwicklungs- und Managementprozesse ermöglichen, denen ein langfristig genutztes und fortgeführtes komplexes Softwaresystem unterliegt.

Mit dem Begriff *Komplexität eines Softwaresystems* beziehen wir uns dabei zum einen auf die Größe der Implementierung des Systems, die bei den meisten Legacy-Systemen mehrere Millionen Lines-of-Code umfasst. Zum anderen bezieht sich die Komplexität auf die Vielzahl der beteiligten Entwickler, die über Jahre in unterschiedlichen Rollen und in unterschiedlichen Entwicklungsphase das System aufbauen und fortführen. Weiter bezieht sich die Komplexität auf die über Jahre andauernde Weiterentwicklung und Überarbeitung existierender Systemkomponenten, in deren Folge sich u. a. konzeptionelle Dissonanzen, Implementierungsredundanzen, Architektur-Asymmetrien und stark variierenden Lösungsqualitäten niederschlagen. Allgemein dient Komplexität bei Softwaresystemen als Maß für den Schwierigkeitsgrad, mit dem das Softwaresystem, seine Bestandteile und die mit ihm verbundenen Prozesse seiner Entwicklung und Nutzung verstehbar, nachvollziehbar und überprüfbar sind.

Mit der Komplexität eines Softwaresystems wächst der Aufwand, der insbesondere mittel- und langfristig notwendig ist, um das System zu warten und fortzuentwickeln. Wartung und Pflege repräsentieren den Großteil der insgesamt anfallenden Herstellungskosten eines Softwaresystems [95], so dass Verfahren und Strategien zur Beherrschung der Komplexität von zentralem Interesse sind, die Herstellungskosten zu senken und die Qualität des Softwaresystems zu verbessern.

1.1.1 Softwaremodelle

Ein wesentlicher Schritt zur Beherrschung der Komplexität von Softwaresystemen stellt der zunehmend konsequente Einsatz von Softwaremodellen dar, wie sie z. B. durch die Unified Modeling Language (UML), definiert sind. „UML helps you specify, visualize, and document models of software systems, including their structure and design […]" [75]. Ein solches Modell existiert nebenläufig zu der Implementierung des Softwaresystems. Um die Konsistenz zwischen Modell und Implementierung zu gewährleisten, muss daher nach einer Modifikation des Modells der Quellcode entsprechend der Modelländerung angepasst werden. Umgekehrt muss nach einer Modifikation des Quellcodes geprüft werden, ob die Implementierung noch immer dem Modell entspricht, wobei gegebenenfalls eine Anpassung des Modells notwendig ist. Diese Anpassung ist teilweise, z. B. für Klassendiagramme, automatisiert möglich. Aktivitätsdiagramme hingegen lassen sich nur durch eine manuelle Modellanpassung exakt fortführen. Eine Gewährleistung, dass das explizite Modell die tatsächliche Implementierung des Softwaresystems beschreibt, ist nur durch ein konsequentes Einhalten des wechselseitigen Zyklus von Quellcodeanpassung und Modellanpassung gegeben.

In der Praxis gestaltet sich die Einhaltung dieses Zyklus schwierig, da insbesondere bei der

Verwendung von agilen Methoden in der Softwareentwicklung [1] wie z. B. dem eXtreme Programming [32] schnelle Iterationszyklen zur Auslieferung funktionierender Software erwünscht sind, um möglichst schnell die Funktionalität der Software auf Kundenwünsche anpassen zu können. Die Angleichung eines nebenläufig zur Implementierung existierenden Modells steht im Hintergrund. Ferner ergeben sich Dissonanzen zwischen expliziten Systemmodellen und der Implementierung im weiteren Verlauf des Lebenszyklus des Softwareproduktes während seiner Weiterentwicklung und der Überarbeitung existierender Systemkomponenten.

Die Gewährleistung, dass insbesondere mit zunehmenden Alter eines Softwaresystems ein aktuelles und vollständiges Systemmodell vorliegt, das widerspruchsfrei zu der Implementierung ist, kann also nicht gegeben werden. Eine Ausnahme besteht bei der Verwendung von Modell-getriebenen Softwareentwicklungs-Ansätzen (Model Driven Architecture MDA) [76], bei denen der Quellcode automatisch aus einem vollständigen Modell generiert wird. Die Entwicklung der Software findet ausschließlich auf der Abstraktionsebene des Modells statt. In der Praxis werden MDA-Verfahren in Großprojekten schon verwendet, allerdings ist die Quellcodegenerierung nicht vollständig, so dass große Teile der Implementierung weiterhin explizit von Entwicklern geschrieben werden müssen (siehe z. B. [89] oder [59]).

Ein verlässliches Verständnis von der Struktur, dem Verhalten, von Abhängigkeiten und Einflüssen eines Softwaresystems und seiner Komponenten kann daher nur auf Basis seiner Implementierung gewonnen werden.

1.1.2 Softwarevisualisierung

Softwarevisualisierung stellt Mittel bereit, die die Verständnisgewinnung über Softwaresysteme unterstützen. Sie umfasst „[...] the development and evaluation of methods for graphically representing different aspects of software, including its structure, its abstract and concrete execution, and its evolution" [84].

In der Softwarevisualisierung können drei Aufgabenbereiche unterschieden werden, in denen die Verständnisgewinnung eine besondere Rolle spielt:

1. Konzeption und Entwicklung neuer Softwaresystemen;
2. Wartung, Erweiterung und Wiederverwendung existierender Softwaresysteme;
3. Management und Monitoring des Software-Entwicklungsprozesses.

Bei der Konzeption und Entwicklung neuer Softwaresysteme finden im Kontext der Objektorientierung Softwarevisualisierungstechniken Einsatz, die auf der Visualisierung von Systemmodellen beruhen. Die Modelle, die in Diagrammform statische Aspekte der Softwarearchitektur, dynamische Aspekte der Systemausführung und physische Aspekte der Systemoperationalisierung umfassen, werden durch die Entwickler erstellt und mit der Implementierung umgesetzt. Sie bleiben nach der Implementierung nur teilweise an diese gekoppelt. Für das Forward-Engineering stehen mit der Unified Modeling Language leistungsstarke, klassische Modellierungs- und Visualisierungsansätze bereit.

Bei der Wartung, Erweiterung und Wiederverwendung hingegen erweisen sich die Methoden aus dem Forward-Engineering als nur bedingt einsetzbar, da im allgemeinen keine explizit entwickelten oder aktuellen Systemmodelle bereitstehen. Das Reverse-Engineering stellt jedoch den Hauptfall des Software-Engineerings in der Zukunft dar [37]. Die damit verbundenen Aufgaben umfassen die funktionale Erweiterung des Systems, die Fehlerbehebung und die Optimierung. Zur Durchführung wird auf unterschiedliche Methoden, wie z. B. das Refactoring, zurückgegriffen. Softwarevisualisierungstechniken, die automatisiert Informationen über Struktur, Abhängigkeiten und Dynamik generieren und vermitteln können, helfen diese Aufgaben zeit- und kosteneffektiv zu bewältigen. Zudem lassen sich mit geeigneten Visualisierungstechniken Systemteile extrahieren, die Hinweise auf ein besonderes Verhalten aufweisen. So können z. B. fehleranfällige oder funktionsüberladende Systemteile leichter identifiziert werden.

Durch die Verknüpfung von Informationen, die aus dem Quellcode gewonnen werden können, mit Informationen, die Aufschluss über die zeitliche Entwicklung des Softwaresystems durch Entwickler geben, lassen sich Visualisierungstechniken für Management- oder Controlling-Aufgaben entwickeln. Hierdurch kann ein Monitoring des Softwareentwicklungsprozesses erfolgen und Entscheidungsunterstützung gegeben werden.

1.2 Grundlagen

In der Literatur wird bei der Softwarevisualisierung zwischen Programm- und Algorithmus-Visualisierung unterschieden [81]. Im folgenden werden unter dem Begriff *Softwarevisualisierung* Visualisierungstechniken aus beiden Bereichen verstanden, die auf die Verständnisgewinnung über die Struktur, das Verhalten und die Evolution von komplexen Softwaresystemen zielen. Wir

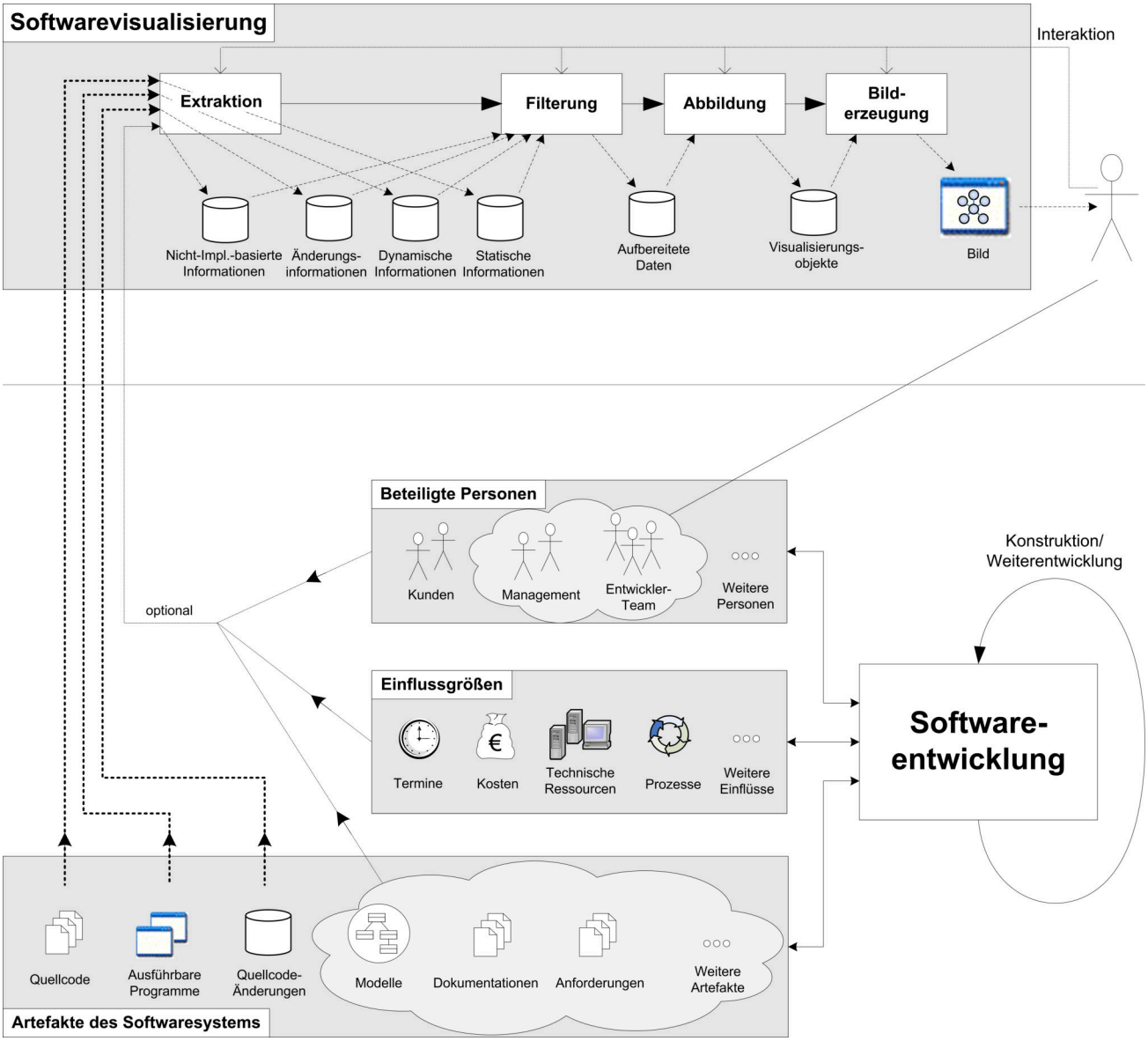

Abbildung 1: Implementierungsbasierte Softwarevisualisierung.

schränken den Begriff der Softwarevisualisierung weiter ein, indem wir Visualisierungen betrachten, die auf Basis der Implementierung eines Softwaresystemen erstellt werden.

1.2.1 Softwarevisualisierung in der Softwareentwicklung

Abbildung 1 skizziert, wie die Softwarevisualisierung in den Prozess der Softwareentwicklung eingebettet ist:

Drei Gruppen von Aspekten spielen bei der Softwareentwicklung eine Rolle:

1) Beteiligte Personengruppen;
2) Einflussgrößen;
3) Artefakte des Softwaresystems.

Die wichtigsten Personengruppen, die an der Konstruktion und Weiterentwicklung von Softwaresystemen beteiligt sind, sind das Entwicklerteam, das Management und die Kunden. Termine, Kosten, technische Ressourcen und Prozesse stellen Einflussgrößen während der Softwareentwicklung dar. Das Softwaresystem selbst besteht aus einer Menge von Artefakten, die während der Entwicklung erzeugt und modifiziert werden:

- Quellcode;
- Ausführbare Programme;
- Informationen über Quellcodeänderungen;
- Explizite Modelle über die Struktur oder das Verhalten des Systems (z. B. in Form eines UML-Modells);
- Dokumentationen;
- Anforderungen;
- Weitere Artefakte.

Softwarevisualisierung zielt auf die Unterstützung der an der Softwareentwicklung beteiligten Personengruppen, insbesondere Entwickler und Mana-

ger, bei der Verständnisgewinnung über das Softwaresystem. Für die Visualisierung sind als Datenquellen implementierungsbezogene Artefakte von besonderem Interesse, da diese untereinander konsistenten Artefakte die Struktur, das Verhalten und die Evolution des Softwaresystems vollständig definieren:

- Quellcode;
- Ausführbare Programme;
- Informationen über Quellcodeänderungen.

Optional können neben den implementierungsbezogenen Artefakten sowohl Informationen über weitere Softwareartefakte als auch Informationen über Einflussgrößen oder beteiligte Personengruppen in die Visualisierung eingehen.

Explizite Systemmodelle oder Systemdokumentationen sind möglicherweise veraltet und weisen Widersprüche zu den implementierungsbezogenen Artefakten auf. Sie stellen daher keine verlässliche Beschreibung des Systems dar. Die Überprüfung dieser Artefakte auf ihre Aktualität hin stellt daher eine interessante Anwendung für die Softwarevisualisierung dar.

Das Verknüpfen von Informationen über implementierungsbezogenen Softwareartefakte mit weiteren Aspekten des Entwicklungsprozesses ist für die Unterstützung von Management-Aufgaben sinnvoll. Visualisierungen dieser Zusammenhänge bieten Entscheidungshilfen für das Management und fördern eine Optimierung des Entwicklungsprozesses, indem sie mögliche Schwachpunkte des Prozesses ersichtlich machen. Problematisch hierbei ist jedoch, dass sich diese zusätzlichen Informationen im Allgemeinen nicht automatisiert gewinnen lassen. Eine Ausnahme bilden Informationen über die Softwareentwicklung, wenn ein Versionsverwaltungssystem genutzt wird. Neben den Änderungen am Quellcode werden Angaben über die für die Änderungen verantwortlichen Entwickler gespeichert. Diese zusätzlichen Informationen können zusammen mit den Quellcodeänderungen vollständig automatisiert vom Visualisierungssystem erfasst werden.

1.2.2 Softwarevisualisierungs-Pipeline

In Abbildung 1 ist ein Modell einer Softwarevisualisierungs-Pipeline dargestellt. Es orientiert sich an dem allgemeinen Modell der Visualisierungspipeline [91]. Sie beinhaltet vier sequentielle Schritte:

1) Extraktion;
2) Filterung;
3) Abbildung;
4) Bilderzeugung.

Nach dem Extraktionsschritt stehen für die Visualisierungs-Pipeline unterschiedliche Typen von Informationen über das Softwaresystem zur Verfügung:

- Informationen, die aus dem Quellcode, extrahiert werden (*statische Informationen*);
- Informationen, die während der Programmausführung erzeugt werden (*dynamische Informationen*);
- Informationen, die über die zeitliche Entwicklung des Softwaresystems im Softwareentwicklungsprozess Aufschluss geben (*Änderungsinformationen*);
- Informationen über weitere Softwareartefakte, Einflussgrößen bei der Entwicklung oder über an der Entwicklung beteiligte Personen (*Nichtimplementierungsbasierte Informationen*).

Im Filterungsschritt werden diese Rohdaten durch Reduktion und Verknüpfung aufbereitet und in einem weiteren Schritt, der Abbildung, Visualisierungsobjekten zugeordnet. Das heißt, es wird ein Modell erzeugt, in dem geometrische Primitive nach Konstruktionsregeln im Darstellungsraum angeordnet werden. Die Primitive können mit Attributen, wie z. B. einer Farbe, versehen sein. In der Bilderzeugung wird in Abhängigkeit von Parametern wie z. B. der Betrachterposition aus dem geometrischen Modell ein Bild generiert, das von den Nutzern betrachtet wird.

Für ein interaktives Explorieren des dargestellten Informationsraums müssen Methoden zur Navigation bereitgestellt werden. Dies geschieht, indem den Nutzern ein Eingreifen in die Bilderzeugung erlaubt wird. Idealerweise können zudem die Extraktion, die Filterung und die Abbildung von den Nutzern konfiguriert werden.

1.3 Softwarevisualisierungs-Werkzeuge

Für die Akzeptanz eines Softwarevisualisierung-Werkzeuges ist es notwendig, dass der Prozess der Visualisierungserzeugung weitgehend automatisiert abläuft. Dies gilt zum einen für die Informationsgewinnung und zum anderen für die Anordnung der grafischen Elemente in der Visualisierung (Layout).

Für die sinnvolle Verwendung eines Softwarevisualisierungs-Werkzeugs ist zudem der Aspekt der Skalierbarkeit zu berücksichtigen. Informationsvisualisierung erreicht unter anderem dadurch einen Mehrwert, dass die menschliche Fähigkeit der Mustererkennung unterstützt wird. Dass heißt,

dass Softwarevisualisierung vor allem bei großen Informationsvolumen einen signifikanten Vorteil gegenüber z. B. der direkten Quellcodebetrachtung bietet. Der Erzeugungs- bzw. Extraktionsschritt von Informationen über ein Softwaresystem durch ein Softwarevisualisierungs-Werkzeug sollte daher große Datenmengen verarbeiten können. Ebenso sollte das Visualisierungsprinzip für große Datenmengen ausgelegt sein. Dabei sollte eine interaktive Exploration der Informationen möglich sein.

Ein weitere Aspekt, der für die Akzeptanz eines Softwarevisualisierungs-Werkzeug eine Rolle spielt, ist die Einbettung desselben in eine Entwicklungsumgebung. Die parallele Verwendung des *stand-alone* Visualisierungs-Werkzeugs bei der Softwareentwicklung bedeutet einen Mehraufwand für die Nutzer und erhöht somit die Hemmschwelle für die Nutzung.

1.4 Implementierungsbasierte Softwarevisualisierung

Ziel des Reports *Konzepte der Softwarevisualisierung für komplexe, objektorientierte Softwaresysteme* ist es, einen Überblick über aktuelle Konzepte im Bereich der Softwarevisualisierung zu geben. Es wird dabei u.a. auf folgende Fragestellungen eingegangen:

- Für welchen Anwendungsbereich ist welches Visualisierungskonzept sinnvoll?
- Wann ergibt sich durch die Visualisierung ein Mehrwert gegenüber anderen Techniken zur Verständnisgewinnung?
- Welche Informationen können visualisiert werden, und wie werden diese erzeugt?
- Wo liegen konzeptionelle bzw. technische Grenzen des jeweiligen Visualisierungsansatzes?
- Wie weit ist der Visualisierungsprozess automatisierbar?

In fünf Arbeitspaketen werden unterschiedliche Aspekte der Softwarevisualisierung untersucht:

- Der Fokus bei der Analyse des Softwarevisualisierungs-Werkzeugs SHriMP/Creole [97] liegt auf der Untersuchung graphbasierter Visualisierungstechniken und speziell der Exploration von hierarchischen Informationsräumen. Die bei SHriMP/Creole visualisierten Informationen sind statische Elemente des UML-Modells.
- Bei der Untersuchung des Softwarevisualisierungs-Werkzeugs CodeCrawler [15] wird auf die Bedeutung von Softwaremetriken bei der Softwarevisualisierung eingegangen.
- Codezeilenbasierte Visualisierungstechniken gehören mit zu den ältesten verwendeten Techniken im Bereich der Softwarevisualisierung und werden noch immer in aktuellen Konzepten genutzt. Die Gründe hierfür werden untersucht. Dabei werden diese Techniken, mit denen Softwaresysteme auf einer niedrigen Abstraktionsebene abgebildet werden, analysiert.
- Die Beschränkung der Informationsdichte bei 2-dimensionalen Darstellungen lässt sich durch eine Erweiterung des Darstellungsraums auf drei Dimensionen umgehen. Bei der Untersuchung von Landschafts- und Stadtmetaphern als Basis für Visualisierungstechniken im 2½-dimensionalen Darstellungsraum wird die Bedeutung der Dimension des Darstellungsraums erörtert und Konzepte zur Softwarevisualisierung basierend auf den genannten Metaphern analysiert.
- Die Visualisierung der Evolution eines Softwaresystems kann z. B. für Management-Aufgaben sinnvoll sein. In der Betrachtung entsprechender Konzepte wird zum einen auf die Entwicklung des Quellcodes und damit der Funktionalität der Softwareartefakte im Laufe der Zeit eingegangen. Zum anderen wird analysiert, wie die arbeitsteilige Entwicklung von Softwaresystemen durch Visualisierung unterstützt werden kann.